Computational Geometry - Homework I

K. Subramani
LCSEE,
West Virginia University,
Morgantown, WV
{ksmani@csee.wvu.edu}

1 Instructions

1. The homework is due on February 28, in class. Each question is worth 5 points.

2. Attempt as many problems as you can. You will be given partial credit, as per the policy discussed in class.

2 Problems

1. **Tail Bounds:** In class, we established that the RANDOMIZED-QUICKSELECT() algorithm runs in expected time $O(n)$, when asked to find the k^{th} largest element in an array of n elements. Argue that there exists a constant c, such that the probability that more than $c \cdot n \cdot \log n$ comparisons are made in a run of RANDOMIZED-QUICKSELECT() is at most $\frac{1}{n}$.

2. **Algorithm Design:** Given a set of n points in the plane, devise an algorithm that runs to check whether there exists a subset of 3 points, which are collinear. Your algorithm should run in time $O(n^2 \cdot \log n)$.

3. **Convex Hulls:**

   (a) Let P be a set of points in the plane. Let P be a convex polygon, whose vertices are points from P and which contains all the points in P. Prove that P is uniquely defined and that it is the intersection of all convex sets containing P.

   (b) Let $p = (p_x, p_y)$ and $q = (q_x, q_y)$ be two points in the plane. We wish to test whether the points $r = (r_x, r_y)$ lies to the left or right of the segment $\bar{pq}$. Using first principles, explain how the sign of the determinant of D can be used for this purpose, where,

   $$D = \begin{bmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{bmatrix}$$

4. **Line Intersection:** Let S be a set of n disjoint segments in the place and let p be any point which does not lie on any segment in S. The goal is to determine all the line segments that are *visible* from p. Note that a segment l in S is visible from p, if there exists a point q on l, such that the segment $\overline{pq}$ intersects only segment l. Devise an algorithm that runs in time $O(n \cdot \log n)$ for this problem.

5. **Backwards Analysis:** Professor Amarsen proposes the following algorithm to find the maximum element in an array of n elements.

```
Function FIND-MAX(A, n)
if (n = 1) then
    return(A[1])
else
    Extract an element randomly from A and call it x. {x is no longer in A.}
    y = FIND-MAX(A, n − 1)
    if (x ≤ y) then
        return(y)
    else
        Compare x with all the remaining elements in A and return the maximum
    end if
end if
```

Algorithm 2.1: Finding Maximum in Paranoid Fashion

Is this algorithm correct? What is the worst-case number of comparisons on a run? How many comparisons are made in the expected case?